UNITED STATES PATENT APPLICATION

for

A MECHANISM TO INCREASE DATA COMPRESSION IN A CACHE

Inventors:

Ali-Reza Adl-Tabatabai
Anwar M. Ghuloum
Eric Sprangle

Prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard
Los Angeles, CA 90025-1026
(303) 740-1980

File No.: 042390.P17411

"Express Mail" mailing label number EL962312095US
Date of Deposit September 30, 2003
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, P.O. Box 1450, Alexandria, VA 22313-1450

Leah Schwenke
(Typed or printed name of person mailing paper or fee)

Leah Schwenke
(Signature of person mailing paper or fee)

A MECHANISM TO INCREASE DATA COMPRESSION IN A CACHE

FIELD OF THE INVENTION

[0001] The present invention relates to computer systems; more particularly, the present invention relates to central processing unit (CPU) caches.

BACKGROUND

[0002] Currently, various methods are employed to compress the content of computer system main memories such as Random Access Memory (RAM). These methods decrease the amount of physical memory space needed to provide the same performance. For instance, if a memory is compressed using a 2:1 ratio, the memory may store twice the amount of data at the same cost, or the same amount of data at half the cost.

[0003] One such method is Memory Expansion Technology (MXT), developed by International Business Machines (IBM) of Armonk, New York. MXT addresses system memory costs with a memory system architecture that doubles the effective capacity of the installed main memory. Logic-intensive compressor and decompressor hardware engines provide the means to simultaneously compress and decompress data as it is moved between the shared cache and the main memory. The compressor encodes data blocks into as compact a result as the algorithm permits.

[0004] However, there is currently no method for compressing data that is stored in a cache. Having the capability to compress cache data would result in similar advantages as main memory compression (e.g., decreasing the amount of cache space needed to provide the same performance).

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The present invention will be understood more fully from the detailed description given below and from the accompanying drawings of various embodiments of the invention. The drawings, however, should not be taken to limit the invention to the specific embodiments, but are for explanation and understanding only.

[0006] **Figure 1** illustrates one embodiment of a computer system;

[0007] **Figure 2** illustrates one embodiment of a physical cache organization;

[0008] **Figure 3** illustrates one embodiment of a logical cache organization;

[0009] **Figure 4A** illustrates an exemplary memory address implemented in an uncompressed cache;

[0010] **Figure 4B** illustrates one embodiment of a memory address implemented in a compressed cache;

[0011] **Figure 5** illustrates one embodiment of a tag array entry for a compressed cache;

[0012] **Figure 6** is a block diagram illustrating one embodiment of a cache controller;

[0013] **Figure 7** illustrates one embodiment of a set and way selection mechanism in a compressed cache;

[0014] **Figure 8** illustrates one embodiment of tag comparison logic;

[0015] **Figure 9** illustrates another embodiment of a tag array entry for a

compressed cache;

[0016] **Figure 10** illustrates another embodiment of tag comparison logic;

[0017] **Figure 11** illustrates one embodiment of byte selection logic;

[0018] **Figure 12** illustrates one embodiment of a pool of bytes cache;

[0019] **Figure 13** illustrates another embodiment of a pool of bytes cache;

[0020] **Figure 14** illustrates yet another embodiment of a pool of bytes cache;

[0021] **Figure 15** illustrates one embodiment of a cache lookup scheme; and

[0022] **Figure 16** illustrates another embodiment of a cache lookup scheme.

DETAILED DESCRIPTION

[0023] A mechanism for compressing data in a cache is described. In the following description, numerous details are set forth. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form, rather than in detail, in order to avoid obscuring the present invention.

[0024] Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

[0025] **Figure 1** is a block diagram of one embodiment of a computer system 100. Computer system 100 includes a central processing unit (CPU) 102 coupled to bus 105. In one embodiment, CPU 102 is a processor in the Pentium® family of processors including the Pentium® II processor family, Pentium® III processors, and Pentium® IV processors available from Intel Corporation of Santa Clara, California. Alternatively, other CPUs may be used.

[0026] A chipset 107 is also coupled to bus 105. Chipset 107 includes a memory control hub (MCH) 110. MCH 110 may include a memory controller 112 that is coupled to a main system memory 115. Main system memory 115

stores data and sequences of instructions and code represented by data signals that may be executed by CPU 102 or any other device included in system 100.

[0027] In one embodiment, main system memory 115 includes dynamic random access memory (DRAM); however, main system memory 115 may be implemented using other memory types. Additional devices may also be coupled to bus 105, such as multiple CPUs and/or multiple system memories.

[0028] In one embodiment, MCH 110 is coupled to an input/output control hub (ICH) 140 via a hub interface. ICH 140 provides an interface to input/output (I/O) devices within computer system 100. For instance, ICH 140 may be coupled to a Peripheral Component Interconnect bus adhering to a Specification Revision 2.1 bus developed by the PCI Special Interest Group of Portland, Oregon.

[0029] According to one embodiment, a cache memory 103 resides within processor 102 and stores data signals that are also stored in memory 115. Cache 103 speeds up memory accesses by processor 103 by taking advantage of its locality of access. In another embodiment, cache 103 resides external to processor 103.

Compressed Cache

[0030] According to one embodiment, cache 103 includes compressed cache lines to enable the storage of additional data within the same amount of area. **Figure 2** illustrates one embodiment of a physical organization for cache 103. In one embodiment, cache 103 is a 512 set, 4-way set associative cache.

However, one of ordinary skill in the art will appreciate that caches implementing other sizes may be implemented without departing from the true scope of the invention.

[0031] A tag is associated with each line of a set. Moreover, a compression bit is associated with each tag. The compression bits indicate whether a respective cache line holds compressed data. When a compression bit is set, the physical memory of the cache line holds two compressed companion lines. Companion lines are two lines with addresses that differ only in the companion bit (e.g., two consecutive memory lines aligned at line alignment).

[0032] In one embodiment, the companion bit is selected so that companion lines are adjacent lines. However, any bit can be selected to be the companion bit. In other embodiments, it may be possible to encode the compression indication with other bits that encode cache line state, such as the MESI state bits, thus eliminating this space overhead altogether.

[0033] When the compression bit is not set, the physical memory of the cache line holds one line uncompressed. Shaded compression bits in **Figure 2** illustrate compressed cache lines. **Figure 3** illustrates one embodiment of a logical organization for cache 103. As shown in **Figure 3**, cache lines are compressed according to a 2:1 compression scheme. For example, the second line of set 0 is compressed, thus storing two cache lines rather than one.

[0034] In one embodiment, each cache line holds 64 bytes of data when not compressed. Thus, each cache line holds 128 bytes of data when compressed.

The effect of the described compression scheme is that each cache tag maps to a variable-length logical cache line. As a result, cache 103 may store twice the amount of data without having to increase in physical size.

[0035] Referring back to **Figure 1**, a cache controller 104 is coupled to cache 103 to manage the operation of cache 103. Particularly, cache controller 104 performs lookup operations of cache 103. According to one embodiment, the hashing function that is used to map addresses to physical sets and ways is modified from that used in typical cache controllers. In one embodiment, the hashing function is organized so that companion lines map to the same set. Consequently, companion lines may be compressed together into a single line (e.g., way) that uses one address tag.

[0036] **Figure 4A** illustrates an exemplary memory address implemented in an uncompressed cache. In a traditional cache, an addressed is divided according to tag, set and offset components. The set component is used to select one of the sets of lines. Similarly, the offset component is the low order bits of the address that are used to select bytes within a line.

[0037] **Figure 4B** illustrates one embodiment of a memory address implemented for lookup in a compressed cache. **Figure 4B** shows the implementation of a companion bit used to map companion lines into the same set. The companion bit is used in instances where a line is not compressed. Accordingly, if a line is not compressed, the companion bit indicates which of the adjacent lines are to be used.

[0038] In a traditional uncompressed cache, the companion bit is a part of the address and is used in set selection to determine whether an address hashes to an odd or even cache set. In one embodiment, the window of address bits that are used for set selection is shifted to the left by one so that the companion bit lies between the set selection and byte offset bits. In this way, companion lines map to the same cache set since the companion bit and set selection bits do not overlap. The companion bit, which now is no longer part of the set selection bits, becomes part of the tag, though the actual tag size does not increase.

[0039] **Figure 5** illustrates one embodiment of a tag array entry for a compressed cache. The tag array entries include the companion bit (e.g., as part of the address tag bits) and a compression bit. The compression bit causes the compressed cache 103 tag to be one bit larger than a traditional uncompressed cache's tag. The compression bit indicates whether a line is compressed.

[0040] Particularly, the compression bit specifies how to deal with the companion bit. If the compression bit indicates a line is compressed, the companion bit is treated as a part of the offset because the line is a compressed pair. If the compression bit indicates no compression, the companion bit is considered as a part of the tag array and ignored as a part of the offset.

[0041] **Figure 6** is a block diagram illustrating one embodiment of cache controller 104. Cache controller 104 includes set and way selection logic 610, byte selection logic 620 and compression logic 630. Set and way selection logic 610 is used to select cache lines within cache 103. **Figure 7** illustrates one embodiment

of set and way selection logic 610 in a compressed cache.

[0042] Referring to **Figure 7**, set and way selection logic 610 includes tag comparison logic 710 that receives input from a tag array to select a cache line based upon a received address. The tag comparison logic 710 takes into account whether a cache line holds compressed data. Because cache lines hold a variable data size, tag comparison logic 710 is also variable length, depending on whether a particular line is compressed or not. Therefore, the tag match takes into account the compression bit.

[0043] **Figure 8** illustrates one embodiment of tag comparison logic 710 includes exclusive-nor (XNOR) gates 1-n, an OR gate and an AND gate. The XNOR gates and the AND gate is included in traditional uncompressed caches, and are used to compare the address with tag entries in the tag array until a match is found. The OR gate is used to select the companion bit depending upon the compression state of a line.

[0044] The companion bit of the address is selectively ignored depending on whether the compression bit is set. As discussed above, if the compression bit is set, the companion bit of the address is ignored during tag match because the cache line contains both companions. If the compression bit is not set, the companion bit of the address is compared with the companion bit of the tag.

[0045] The "Product of XNOR" organization of the equality operator, therefore, uses the OR gate to selectively ignore the companion bit. In one embodiment, because the tag's companion bit is ignored when the compression

bit is set (e.g., it is a "don't care"), the tag's companion bit can be used for other uses. For example, when a line is compressed, this bit may be used as a compression format bit to select between two different compression algorithms. In another example, the companion bit can be used to encode the ordering of companion lines in the compressed line

[0046] In other embodiments, each cache line is partitioned into two sectors that are stored in the same physical cache line only if the sectors can be compressed together. In the tag entry, the companion and compression bits become sector presence indications, as illustrated in **Figure 9**. In this embodiment, the companion bit is a sector identifier (e.g., upper or lower) and thus has been relabeled as sector ID.

[0047] Accordingly, a "01" indicates a lower sector (not compressed), "10" indicates an upper sector (not compressed), and a "11" indicates both sectors (2:1 compression). Also, in this arrangement the physical cache line size is the same as the logical sector size. When uncompressed, each sector of a line is stored in a different physical line within the same set (e.g., different ways of the same set).

[0048] When compressible by at least 2:1, the two sectors of each line are stored in a single physical cache line (e.g., in one way). It is important to note that this differs from traditional sectored cache designs in that different logical sectors of a given logical line may be stored simultaneously in different ways when uncompressed.

[0049] In one embodiment, a free encoding ("00") is used to indicate an

invalid entry, potentially reducing the tag bit cost if combined with other bits that encode the MESI state. Because this is simply an alternative encoding, the sector presence bits require slightly difference logic to detect tag match. **Figure 10** illustrates another embodiment of tag comparison logic 610 implementing sector presence encoding.

[0050] Referring back to **Figure 6**, byte selection logic 620 selects the addressed datum within a line. According to one embodiment, byte selection logic 620 depends on the compression bit. **Figure 11** illustrates one embodiment of byte selection logic 620. Byte selection logic 620 includes a decompressor 1110 to decompress a selected cache line if necessary. An input multiplexer selects between a decompressed cache line and an uncompressed cache line depending upon the compression bit.

[0051] In one embodiment, the range of the offset depends on whether the line is compressed. If the line is compressed, the companion bit of the address is used as the high order bit of the offset. If the line is not compressed, decompressor 1110 is bypassed and the companion bit of the address is not used for the offset. The selected line is held in a buffer whose size is twice the physical line size to accommodate compressed data.

[0052] Alternative embodiments may choose to use the companion bit to select which half of the decompressed word to store in a buffer whose length is the same as the physical line size. However, buffering the entire line is convenient for modifying and recompressing data after writes to the cache.

[0053] Referring back to **Figure 6**, compression logic 630 is used to compress cache lines. In one embodiment, cache lines are compressed according to a Lempel-Ziv compression algorithm. However in other embodiments, other compression algorithms (e.g., WK, X-Match, sign-bit compression, run-length compression, etc.) may be used to compress cache lines.

[0054] Compression logic 630 may also be used to determine when a line is to be compressed. According to one embodiment, opportunistic compression is used to determine when a line is to be compressed.

In opportunistic compression, when a cache miss occurs the demanded cache line is fetched from memory 115 and cache 103 attempts to compress both companions into one line if its companion line is resident in the cache. If the companion line is not resident in cache 103 or if the two companions are not compressible by 2:1, then cache 103 uses its standard replacement algorithm to make space for the fetched line.

[0055] Otherwise, cache 103 reuses the resident companion's cache line to store the newly compressed pair of companions thus avoiding a replacement. Note, that it is easy to modify the tag match operator to check whether the companion line is resident without doing a second cache access. For example, if all of the address tag bits except for the companion bit match, then the companion line is resident.

[0056] In another embodiment, a prefetch mechanism is used to determine if lines are to be compressed. In the prefetch compression mechanism the

opportunistic approach is refined by adding prefetching. If the companion of the demand-fetched line is not resident, the cache prefetches the companion and attempts to compress both companions into one line.

[0057] If the two companion lines are not compressible by 2:1, cache 103 has the choice of either discarding the prefetched line (thus wasting bus bandwidth) or storing the uncompressed prefetched line in the cache (thus potentially resulting in a total of two lines to be replaced in the set). In one embodiment, the hardware can adaptively switch between these policies based on how much spatial locality and latency tolerance the program exhibits.

[0058] In another embodiment, a victim compression mechanism is used to determine if lines are to be compressed. For victim compression, there is an attempt to compress a line that is about to be evicted (e.g., a victim). If a victim is not already compressed and its companion is resident, cache 103 gives the victim a chance to remain resident in the cache by attempting to compress it with its companion. If the victim is already compressed, its companion is not resident, or the victim and its companion are not compressible by 2:1, the victim is then evicted. Otherwise, cache 103 reuses the resident companion's cache line to store the compressed pair of companions, thus avoiding the eviction.

[0059] As data is written, the compressibility of a line may change. A write to a compressed pair of companions may cause the pair to be no longer compressible. Three approaches may be taken if a compressed cache line becomes uncompressible. The first approach is to simply evict another line to

make room for the extra line resulting from the expansion. This may cause two companion lines to be evicted if all lines in the set are compressed.

[0060] The second approach is to evict the companion of the line that was written. The third approach is to evict the line that was written. The choice of which of these approaches to take depends partly on the interaction between the compressed cache 103 and the next cache closest to the processor (e.g., if the L3 is a compressed cache then it depends on the interaction between L3 and L2).

[0061] Assuming that the compressed cache is an inclusive L3 cache and that L2 is a write-back cache, the first two approaches include an invalidation of the evicted line in the L2 cache to maintain multi-level inclusion, which has the risk of evicting a recently accessed cache line in L2 or L1. The third approach does not require L2 invalidation and does not have the risk of evicting a recently accessed cache line from L2 because the line that is being written is being evicted from L2.

[0062] The above-described mechanism allows any two cache lines that map to the same set and that differ only in their companion bit to be compressed together into one cache line. In one embodiment, the mechanism modifies the set mapping function and selects the companion bit such that it allows adjacent memory lines to be compressed together, which takes advantage of spatial locality.

Increasing Cache Compressibility

[0063] The above-described cache compression mechanism typically involves some integral factor to compress data. For example, data is compressed by 2:1, 3:1, 4:1 factors. The motivation for the integral factor is to simplify tag matching (e.g., to use a match on tag bits, rather than an arithmetic range check). Matching tag bits entails the tag-addressable contents of the cache corresponding to some power of two number of bytes.

[0064] Further, typical cache line sizes are influenced by addressing constraints to also include some power of two number of bytes. Usually, cache line size and tag-addressable line size are equal. In the case of a compressed cache, line size and tag-addressable line size may not match. However, simplifying tag matching, and a desire to minimize addressing constraints results in the integral compressibility constraint.

[0065] Restricting compressibility to 50% (2:1) compression is a strategy, which penalizes those pairs of lines that are compressible by 49% and 5% equally. In other words, both cases will be treated as uncompressible. Adding 10 bytes per line allows the compression of 90% of resident cache lines. According to one embodiment, additional bytes are added for each cache line, resulting in the capability of compressing a line that are not quite 50% (or 33%, 25%, etc.) compressible.

[0066] In a further embodiment, additional bytes are available on demand from a separate pool. Thus, obviating any requirement of extending each

physical cache line to accommodate some number of extra bytes.

[0067] **Figure 12** illustrates one embodiment of a pool of bytes cache. Each pool of bytes cache is a smaller cache that hold additional bytes for lines that are to be compressed, but does not have sufficient space to compress. In one embodiment, the pool of bytes has a fixed width of multiple bytes. In addition, a pool is allocated to ways of each set. For instance, cache set 0, cache set 1, and so on, each include an allocated pool.

[0068] In another embodiment, a way indicator is associated with every line of each extra bytes pool. The way indicator points to the way to which a particular extra byte field is assigned. Note that no more than two ways are required for each set since not every cache line in a set will need additional bytes. The pool of bytes scheme disclosed in **Figure 12** has the advantage of requiring only one additional lookup in the byte pool per line.

[0069] **Figure 13** illustrates another embodiment of a pool of bytes cache. In this embodiment, the width of the pools are fixed (tough finer grained, e.g. one byte), and many pool entries may map to each way. For example, both bytes in set 1 map to way 3 of the cache. In one embodiment, the ordering of bytes is handled so that each byte field mapped to a particular set is sorted accordingly with respect to the logical ordering in the extended cache line. In this way, lookups should proceed serially through the byte pool finding matches in order until no further matches are found.

[0070] In a further embodiment, an associated LRU state for each pool

entry is inherited from the owning way. Note that a replacement in the main cache may displace additional lines. In a one-to-one byte field mapping, an additional line may be displaced if the byte pool entry is required by a new line. As such, the replacement policy considers both the length of the extended line with the LRU state so that multiple lines are not replaced.

[0071] Moreover, a many-to-one byte field mapping is further complicated by the variable length nature of each extended lines byte pool allocation. Multiple lines may need to be displaced. In addition, in the many-to-one mapping case, the replacement policy ensures that all byte entries mapping to the same set are discarded when one is discarded.

[0072] **Figure 14** illustrates another embodiment of a pool of bytes cache. In this embodiment, a pool of bytes may be shared amongst multiple sets. As shown, for example, sets 0 and 1 share a pool. According to one embodiment, a different hashing function for set mapping into the extra byte pool may be accommodated. The simplest hashing modification is to divide the cache's set count by a power of two so that the set hashing for byte pool is a subset of the bits used for the main cache's set lookup. In one embodiment, a set indication is implemented, instead of a way indication to determine ownership. In a further embodiment, the tag is to be stored.

[0073] **Figure 15** illustrates one embodiment of a parallel cache lookup scheme. The parallel lookup scheme may be used for the one-to-one mapping allocation policy. In a parallel lookup, set bits are simultaneously dispatched to

the main cache and the byte pool cache. The results are appended together and simultaneously fed into the decompression logic.

[0074] **Figure 16** illustrates one embodiment of a serial cache lookup scheme. The serial lookup scheme may be used for each of the mapping allocation policies. For the serial lookup, the first byte pool cache match can be overlapped with the main cache lookup. Subsequent matches are serialized due to the dependence of placement in the extended line on lookup order.

[0075] The pool of bytes implementation increases the potential compressibility of cache contents.

[0076] Whereas many alterations and modifications of the present invention will no doubt become apparent to a person of ordinary skill in the art after having read the foregoing description, it is to be understood that any particular embodiment shown and described by way of illustration is in no way intended to be considered limiting. Therefore, references to details of various embodiments are not intended to limit the scope of the claims which in themselves recite only those features regarded as the invention.